

Arranging Your Virtual Network on FreeBSD

by Michael Gmelin

Modern FreeBSD offers a range of virtualization options, from the traditional jail environment sharing the network stack with the host operating system, over `vnet` jails, which allow each jail to have its own network stack, to `bhyve` virtual machines running their own kernels/operating systems.

Depending on individual requirements, there are different ways to configure the virtual network. Jail and VM management tools can ease the process by abstracting away (at least some of) the underlying complexities.

Document Conventions

This article is based on FreeBSD 12.1-RELEASE, the latest release version of FreeBSD at the time of writing. It assumes that ZFS is used on hosts running jails and `bhyve` VMs.

Unless noted otherwise, packages used in examples are from FreeBSD's quarterly branch 2019Q4 (<https://svnweb.freebsd.org/ports/branches/2019Q4/>). Using the quarterly package repository is the default configuration on a freshly installed FreeBSD system.

All examples are limited to IPv4 for the sake of simplicity.

This article introduces various jail and VM management tools, even though everything shown could be configured directly without installing any packages by modifying system configuration files like `/etc/jail.conf` manually.

Code and terminal interaction/output is formatted **like this**. If you would like to copy and paste code you see in this article, you may do so at <https://blog.grem.de/ayvn>.

Sometimes (but not always) the output of commands is shown in examples for clarity.

Man page references are formatted in *italics*, specifying the manual section in parentheses; e.g., `security(7)` refers to the `security` man page, which can be displayed by typing `man 7 security`, or `man security`, as there are no other man pages of that name in the manual.

FreeBSD package references are formatted in *italics*, using the port's origin, which consists of `<category>/<name>`, e.g., `security/sudo`, which can be installed as a binary package (`pkg install sudo`) or from ports (`cd /usr/ports/security/sudo && make install clean`).

License

This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0) (<https://creativecommons.org/licenses/by/4.0/>).

Plain Jails

Plain (as in *non-VNET*) jails share the network stack with the jailhost they're running on. Therefore, network configuration and firewalling are done on the jailhost and not inside of the jail. They are the traditional way of creating jails and despite their limitations still very useful when it comes to containing software, filesystems, and services. E.g., `ports-mgmt/poudriere`, FreeBSD's bulk package builder and port tester, makes heavy use of plain jails.

Plain Jails Using Inherited IP Configuration

The most basic option is running a jail that inherits its network configuration (all interfaces/IP addresses) from the jailhost. This is a common setup if the jail is mostly used as a container to keep the jailhost (the host the jails run on) "clean" of dependencies. This way the host only requires a minimal number of packages (basics like `security/sudo`, `shells/bash`, and `sysutils/tmux`), while application jails can be snapshotted, backed up, and managed/moved separately.

Example of how to configure, run, and destroy a simple jail using `sysutils/iocage`:

```
root@jailhost:~ # pkg install py36-iocage
...
root@jailhost:~ # iocage activate zroot
ZFS pool 'zroot' successfully activated.
root@jailhost:~ # iocage create -r 12.1-RELEASE -n simplecage ip4=inherit
...
simplecage successfully created!
root@jailhost:~ # iocage console -f simplecage
...
root@simplecage:~ # fetch -q -o - http://canhazip.com
<your public facing IP shown here>
root@simplecage:~ # logout
root@jailhost:~ # iocage destroy -f simplecage
...
Stopping simplecage
Destroying simplecage
root@jailhost:~ #
```

Note: This inherits all interfaces and copies the resolver configuration from the jailhost.

Same example using *sysutils/pot*, an alternative jail manager that aims to provide container-like features and *sysutils/nomad* integration:

```
root@jailhost:~ # pkg install pot
...
root@jailhost:~ # pot init
...
root@jailhost:~ # pot create-base -r 12.1
...
root@jailhost:~ # pot create -p simplepot -b 12.1
...
root@jailhost:~ # pot run simplepot
root@simplepot:~ # fetch -q -o - http://canhazip.com
<your public facing IP shown here>
root@simplepot:~ # exit
root@jailhost:~ # pot destroy -Fp simplepot
==> Destroying pot simplepot
root@jailhost:~ #
```

Note: sysutils/pot is an evolving project, so you might want to grab the version from FreeBSD's latest branch by modifying `etc/pkg/FreeBSD.conf` or install it from ports (`usr/ports/sysutils/pot`). Make sure to set `POT_EXTIF` in `usr/local/etc/pot/pot.cfg` in case your network interface name isn't "em0".

Plain Jails Using a Dedicated IP Address

In case a bit more separation is wanted, a dedicated static IP address can be assigned. This prevents a jail from listening to ports used by the jailhost, e.g., permitting to *ssh(1)* directly into a jail listening to the standard port (22).

In the example below, the jailhost uses 192.168.0.2 as its primary IP address, 192.168.0.1 as the default gateway, and 192.168.0.3 is added as an additional IP address to be used by the jail. The goal is to create a jail on its dedicated static IP address on the local network and run *sshd(8)* inside.

This assumes that the jailhost has its secure shell daemon configured to listen only to the relevant IP address, by setting `ListenAddress` to 192.168.0.2 in `/etc/ssh/sshd_config` and reloading it by running `service sshd reload`.

Static LAN IP jail example using *sysutils/pot*:

```
root@jailhost:~ # pot create -p aliaspot -b 12.1 -N alias -i 192.168.0.3
==> Creating a new pot
==> pot name : aliaspot
==> type : multi
==> base : 12.1
==> pot_base :
==> level : 1
==> network-type: alias
==> ip : 192.168.0.3
==> dns : inherit
root@jailhost:~ # pot run aliaspot
```

Continued

```
...
root@aliaspot:~ # service sshd enable
sshd enabled in /etc/rc.conf
root@aliaspot:~ # service sshd start
...
root@aliaspot:~ # exit
root@jailhost:~ # sockstat -4lj aliaspot
USER      COMMAND  PID    FD PROTO  LOCAL ADDRESS      FOREIGN ADDRESS
root      sshd     7324   3  tcp4   192.168.0.3:22     *:
root      sendmail 7183   3  tcp4   192.168.0.3:25     *:
root      syslogd  7102   5  udp4   192.168.0.3:514    *:
root@jailhost:~ # pot destroy -Fp aliaspot
==> Destroying pot aliaspot
root@jailhost:~ #
```

You can run *ifconfig(8)* in various stages of the provisioning process to understand when the IP alias (192.168.0.33/32) is added/removed to/from the interface. Depending on the use case, it might also make sense to add aliases permanently in the jailhost's */etc/rc.conf*.

*Note: In such a static single IP configuration, the jail's only IP address also (magically) serves as localhost, which can be confusing at times and also means that services that usually listen to localhost and therefore are not reachable from the outside world (such as *sendmail_submit* in the example above) are suddenly exposed. So, it's important to firewall services on the jailhost properly in such a setup (following the best practice of blocking all traffic by default). It's possible to add unique loopback addresses (like 127.0.0.2/8) to a jail, but given the additional complexity this introduces, doing so is only worth the effort in a limited number of use cases.*

Plain Jails Using a VLAN IP Address

A variation of the previous setup is to configure the jail to listen to IP addresses on a dedicated (private) network, either by using a dedicated interface or by adding VLAN interfaces to an existing physical interface. This provides a better segmentation of the network and allows to deploy central filtering, outbound network address translation (NAT), and inbound redirection (DNAT) of network traffic.

In the example below, a VLAN interface (VLAN tag 101, 10.1.1.1/24) is configured on the jailhost (interface "em0") and put to use in a jail configured using *sysutils/iocage*:

```
root@jailhost:~ # sysrc ifconfig_em0_101="10.1.1.1/24"
ifconfig_em0_101: -> 10.1.1.1/24
root@jailhost:~ # ifconfig em0.101 create
root@jailhost:~ # iocage create \
  -r 12.1-RELEASE -n vlancage ip4_addr="em0.101|10.1.1.2"
...
vlancage successfully created!
root@jailhost:~ # iocage console -f vlancage
...
```

Continues next page

Continued

```
root@v lancage:~ # ifconfig -g vlan
em0.101
root@v lancage:~ # ifconfig em0.101
em0.101: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 \
mtu 1500
    options=3<RXCSUM, TXCSUM>
    ether f4:4d:30:aa:bb:cc
    inet 10.1.1.2 netmask 0xffffffff broadcast 10.1.1.2
    groups: vlan
    vlan: 101 vlanpcp: 0 parent interface: em0
    media: Ethernet autoselect (100baseTX <full-duplex>)
    status: active
root@v lancage:~ # logout
root@jailhost:~ # iocage destroy -f v lancage
...
Stopping v lancage
Destroying v lancage
root@jailhost:~ #
```

This assumes that firewalling and NAT happen on a different host/appliance on your network. As the host already has an IP address configured (10.1.1.1/24), adding the jail's IP address as an alias host address (netmask /32) is exactly what was wanted. This also helps configuration-wise, as it allows things like configuring static routes before jails are started and setting up firewall rules based on the interface's network configuration.

In case the jail should use the interface's primary IP address, it's important to configure the IP address and netmask exactly like they're configured on the VLAN interface. In the example above this would mean setting `ip4_addr="em0.101 | 10.1.1.1/24"`. `iocage(8)` is smart enough to not remove the jail's IP address from an interface if it was already configured prior to starting the jail.

Hint: Like most network interfaces, it's possible to name VLAN interfaces semantically; see `rc.conf(5)` (`man rc.conf`) for details.

Plain Jails Using a Loopback IP Address

Under some circumstances, e.g., if the number of IP addresses you can assign to your LAN interface is limited and/or all you are maintaining is a single jailhost, it might make sense to have the jail listen to a local loopback interface. In this case you usually use a local firewall to NAT outbound traffic and redirect inbound traffic (if necessary).

In the `sysutils/iocage` example below, a dedicated local interface named `lo1`, holding the IP address 172.31.255.17/32, is created to serve the jail. In this example the IP address is already assigned by the jailhost on boot and not by `sysutils/iocage` when starting the jail:

```
root@jailhost:~ # sysrc cloned_interfaces+="lo1"
cloned_interfaces: -> lo1
root@jailhost:~ # sysrc ifconfig_lo1="172.31.255.17/32"
```

Continued

```
ifconfig_lo1: -> 172.31.255.17/32
root@jailhost:~ # ifconfig lo1 create
root@jailhost:~ # ifconfig lo1
lo1: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
      options=680003<RXCSUM,TXCSUM,LINKSTATE,RXCSUM_IPV6,TXCSUM_IPV6>
      inet 172.31.255.17 netmask 0xffffffff
      groups: lo
      nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
root@jailhost:~ # iocage create \
  -r 12.1-RELEASE -n locage ip4_addr="lo1|172.31.255.17/32"
...
locage successfully created!
root@jailhost:~ # iocage console -f locage
...
root@locage:~ # host freebsd.org
;; connection timed out; no servers could be reached
root@locage:~ # logout
root@jailhost:~ #
```

Adding Outbound NAT for Public Traffic

At this point the jail exists but has no way to communicate with the outside world. This can be fixed by adding an outbound NAT rule to the firewall.

Warning: Enabling/configuring a firewall is a great way to lock yourself out of a machine. Make sure you have a plan in place on how to regain access in case this happens to you accidentally.

In this example the *pf(4)* (packet filter) firewall is used. It is assumed that there is no firewall configured yet.

The following */etc/pf.conf* is created:

```
ext_if="em0"
www_jail="172.31.255.17"

# outbound NAT for www jail on jailhost's primary IP address
nat on $ext_if from $www_jail -> $ext_if:0

# redirect www traffic into jail
rdr on $ext_if proto tcp to $ext_if:0 port www -> $www_jail

# don't interfere with jailhost's localloop
set skip on lo0

# prevent spoofing
antispoof for lo0
antispoof for lo1
antispoof for $ext_if
```

Continues next page

Nov/Dec 2019

13

```
Continued # block all by default (best practice)
block

# allow any traffic from jail that doesn't go to the jailhost
# (includes traffic inside of the jail to itself)
pass from $www_jail to !$ext_if:0

# allow access to web server
pass proto tcp to $www_jail port www

# allow access to manage host
pass in on $ext_if proto tcp to $ext_if:0 port ssh

# allow outbound traffic
pass out on $ext_if
```

Next, *pf(4)* is enabled and outbound connectivity is verified to work as expected:

```
root@jailhost:~ # service pf enable
pf enabled in /etc/rc.conf
root@jailhost:~ # service pf start
... (session drops) ...
root@jailhost:~ # iocage console -f locage
root@locage:~ # fetch -q -o - http://canhazip.com
<your public facing IP shown here>
root@locage:~ # logout
root@jailhost:~ #
```

Running a Service and Redirecting Traffic to It

Now that there is outbound connectivity within the jail, let's install *www/nginx* inside to serve static content. The necessary firewall rules are already in place (check the *rdr* and *pass* rules that were configured earlier):

```
root@jailhost:~ # grep -B1 "port www" /etc/pf.conf
# redirect www traffic into jail
rdr on $ext_if proto tcp to $ext_if:0 port www -> $www_jail
--
# allow access to web server
pass proto tcp to $www_jail port www
root@jailhost:~ # iocage console -f locage
root@locage:~ # pkg install nginx
...
root@locage:~ # rm /usr/local/www/nginx
root@locage:~ # mkdir /usr/local/www/nginx
root@locage:~ # echo "Hello Jail" >/usr/local/www/nginx/index.html
root@locage:~ # service nginx enable
nginx enabled in /etc/rc.conf
root@locage:~ # service nginx start
Performing sanity check on nginx configuration:
nginx: the configuration file /usr/local/etc/nginx/nginx.conf syntax is ok
```

Continued

```
nginx: configuration file /usr/local/etc/nginx/nginx.conf test is\
successful
Starting nginx.
root@locage:~ # logout
root@jailhost:~ # fetch -q -o - http://172.31.255.17
Hello Jail
root@jailhost:~ #
```

Accessing the web server hosted within the jail from the outside should work at this point and can be verified by pointing a web browser to the server's primary IP address.

Warning: Even though it's possible to create clean and easy to understand configurations this way, these setups don't scale very well and can get quite a hassle to maintain. Also, even though it's possible to firewall between the various jails to a certain extent, it's not very practical and therefore VNET jails should be preferred.

VNET Jails and bhyve VMs

VNET(9)—the network subsystem virtualization infrastructure—is a technology to virtualize the network stack.

Even though VNET first appeared in FreeBSD 8.0, it was considered an experimental feature until recently. With the arrival of FreeBSD 12 it is finally available to everyone without the burden of building a custom kernel.

When applied to jails, VNET allows each jail to have its own network stack. This solves a couple of the shortcomings we've seen so far and gets us improvements like better segregation and isolation, a "normal" local loopback interface, and the possibility to run an independent firewall inside a jail.

VNET Jails Using *sysutils/pot*

This section will demonstrate how to create multiple VNET jails using *sysutils/pot*. Before demonstrating how this is done, the configuration changes to *pf(4)* done above are reverted (in case you didn't run these examples and *pf(4)* isn't running, please replace `service pf reload` with `service pf start`):

```
root@jailhost:~ # rm -f /etc/pf.conf
root@jailhost:~ # pot init
...
Please, check that your PF configuration file is still valid!
root@jailhost:~ # cat /etc/pf.conf
nat-anchor pot-nat
rdr-anchor "pot-rdr/*"
root@jailhost:~ # service pf enable
root@jailhost:~ # service pf reload
```

pot automatically uses VNET when a jail is created with a network-type of *public-bridge*. Based on the "Internal Virtual Network configuration" in

/usr/local/etc/pot/pot.cfg it creates a bridge interface and assigns an IP address that serves as the default gateway for jails to it.

```
root@jailhost:~ # pot vnet-start
pfctl: pf already enabled
root@jailhost:~ # ifconfig bridge0
bridge0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0\
mtu 1500
    ether 02:0e:35:b7:6d:00
    inet 10.192.0.1 netmask 0xffc00000 broadcast 10.255.255.255
    id 00:00:00:00:00:00 priority 32768 hellotime 2 fwddelay 15
    maxage 20 holdcnt 6 proto rstp maxaddr 2000 timeout 1200
    root id 00:00:00:00:00:00 priority 32768 ifcost 0 port 0
    groups: bridge
    nd6 options=1<PERFORMNUD>
```

On jail creation, pot automatically assigns a static IP address from the configured range. On jail start, an *epair(4)* interface is created and its "a-side" is added to the bridge:

```
root@jailhost:~ # pot create -b 12.1 -N public-bridge -p vnetpot1
...
root@jailhost:~ # pot create -b 12.1 -N public-bridge -p vnetpot2
...
root@jailhost:~ # pot start vnetpot1
...
root@jailhost:~ # pot start vnetpot2
...
root@jailhost:~ # pot list
pot name : base-12_1
    network : inherit
    active : false

pot name : vnetpot1
    network : public-bridge
    ip : 10.192.0.3
    active : true

pot name : vnetpot2
    network : public-bridge
    ip : 10.192.0.4
    active : true
root@jailhost:~ # ifconfig bridge0
bridge0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0\
mtu 1500
    ether 02:0e:35:b7:6d:00
    inet 10.192.0.1 netmask 0xffc00000 broadcast 10.255.255.255
    id 00:00:00:00:00:00 priority 32768 hellotime 2 fwddelay 15
    maxage 20 holdcnt 6 proto rstp maxaddr 2000 timeout 1200
    root id 00:00:00:00:00:00 priority 32768 ifcost 0 port 0
    member: epair1a flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
        ifmaxaddr 0 port 7 priority 128 path cost 2000
```

```

    member: epair0a flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
           ifmaxaddr 0 port 6 priority 128 path cost 2000
    groups: bridge
    nd6 options=1<PERFORMNUD>
root@jailhost:~ # ifconfig -g epair
epair0a
epair1a
root@jailhost:~ # ifconfig epair0a
epair0a: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST>\
metric 0 mtu 1500
    options=8<VLAN_MTU>
    ether 02:f5:dd:bc:cd:0a
    groups: epair
    media: Ethernet 10Gbase-T (10Gbase-T <full-duplex>)
    status: active
    nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
root@jailhost:~ # ifconfig epair1a
epair1a: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST>\
metric 0 mtu 1500
    options=8<VLAN_MTU>
    ether 02:d8:52:a6:86:0a
    groups: epair
    media: Ethernet 10Gbase-T (10Gbase-T <full-duplex>)
    status: active
    nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>

```

The *epair(4)* interface's "b-side" is used by the jail, which sets the configured static IP address as configured in its */etc/rc.conf*:

```

root@jailhost:~ # jexec vnetpot1 grep ifconfig /etc/rc.conf
ifconfig_epair0b="inet 10.192.0.3 netmask 255.192.0.0
root@jailhost:~ # jexec vnetpot1 ifconfig epair0b
epair0b: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0\
mtu 1500
    options=8<VLAN_MTU>
    ether 02:f5:dd:bc:cd:0b
    inet 10.192.0.3 netmask 0xffc00000 broadcast 10.255.255.255
    groups: epair
    media: Ethernet 10Gbase-T (10Gbase-T <full-duplex>)
    status: active
    nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
root@jailhost:~ # jexec vnetpot2 grep ifconfig /etc/rc.conf
ifconfig_epair1b="inet 10.192.0.4 netmask 255.192.0.0"
root@jailhost:~ # jexec vnetpot2 ifconfig epair1b
epair1b: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0\
mtu 1500
    options=8<VLAN_MTU>
    ether 02:d8:52:a6:86:0b
    inet 10.192.0.4 netmask 0xffc00000 broadcast 10.255.255.255
    groups: epair
    media: Ethernet 10Gbase-T (10Gbase-T <full-duplex>)
    status: active
    nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>

```



Using the anchors configured in */etc/pf.conf*, pot adds NAT rules to *pf(4)*, which can be inspected using *pfctl(8)*:

```
root@jailhost:~ # pfctl -s nat -a pot-nat
nat on em0 inet from 10.192.0.0/10 to any -> (em0) round-robin
```



Like in the plain jail redirect example above, we want one of the VNET jails to run a web server and make it available to the outside world through an inbound redirect (DNAT). Let's install *www/nginx* inside of the first jail and use pot's "export-port" feature to create the inbound redirect:

```
root@jailhost:~ # pot run vnetpot1
root@vnetpot1:~ # pkg install nginx
...
root@vnetpot1:~ # rm /usr/local/www/nginx
root@vnetpot1:~ # mkdir /usr/local/www/nginx
root@vnetpot1:~ # echo "Hello Jail" >/usr/local/www/nginx/index.html
root@vnetpot1:~ # service nginx enable
nginx enabled in /etc/rc.conf
root@vnetpot1:~ # service nginx start
...
root@vnetpot1:~ # exit
root@jailhost:~ # fetch -q -o - http://10.192.0.3
Hello Jail
root@jailhost:~ # pot export-ports -p vnetpot1 -e 80:80
root@jailhost:~ # pot stop vnetpot1
root@jailhost:~ # pot start vnetpot1
...
root@jailhost:~ # pot show
pot vnetpot1
      disk usage      : 55.2M
====> runtime memory usage require rctl enabled

      Network port redirection
              192.168.0.2 port 80 -> 10.192.0.3 port 80
pot vnetpot2
      disk usage      : 308K
====> runtime memory usage require rctl enabled
root@jailhost:~ #
```



pot creates an anchor containing *redirect pass* rules for exported ports in *pf(4)*, which again can be inspected using *pfctl(8)*:

```
root@jailhost:~ # pfctl -a pot-rdr -s Anchors
      pot-rdr/vnetpot1
root@jailhost:~ # pfctl -a pot-rdr/vnetpot1 -s nat
rdr pass on em0 inet proto tcp \
      from any to 192.168.0.2 port = http -> 10.192.0.3 port 80
```

Note: At this point pf(4) is only used for NATting and redirecting traffic; it does not block any traffic.

VNET Jails Using *sysutils/iocage*

Compared to *sysutils/pot*, *sysutils/iocage* offers more flexibility and fine-grained control when creating custom setups.

Managing Bridges

iocage(8) expects the user to manage the bridges VNET jails connect to, so usually this is done manually, using *ifconfig(8)* and modifying */etc/rc.conf*.

An alternative way to manage bridges is to use *sysutils/vm-bhyve*, which abstracts them as “virtual switches”. As *vm-bhyve(8)* is a great tool to manage bhyve VMs, we’ll take advantage of it to manage bridges for *iocage(8)* jails and later connect bhyve VMs to them.

```
root@jailhost:~ # pkg install vm-bhyve
vm enabled in /etc/rc.conf
root@jailhost:~ # service vm enable
vm enabled in /etc/rc.conf
root@jailhost:~ # sysrc vm_dir=zfs:zroot/vms
vm_dir:  -> zfs:zroot/vms
root@jailhost:~ # zfs create zroot/vms
root@jailhost:~ # vm init
root@jailhost:~ # vm help | grep switch
...
```

Note: vm-bhyve(8) supports different switch types. We limit ourselves to the default “standard” for the time being.

Running `rcorder /usr/local/etc/rc.d/*` shows that *vm* is executed prior to *iocage* on system startup. This means that bridges will be available when jails are connected to them.

With that sorted out, we create a first switch:

```
root@jailhost:~ # vm switch create -a 10.1.1.1/24 services
root@jailhost:~ # vm switch list
NAME      TYPE      IFACE      ADDRESS      PRIVATE  MTU  VLAN  PORTS
services  standard  vm-services 10.1.1.1/24  no      -    -    -
root@jailhost:~ # ifconfig vm-services
vm-services: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0\
mtu 1500
    ether f6:b5:a8:80:78:5e
    inet 10.1.1.1 netmask 0xfffff00 broadcast 10.1.1.255
    id 00:00:00:00:00:00 priority 32768 hellotime 2 fwddelay 15
    maxage 20 holdcnt 6 proto rstp maxaddr 2000 timeout 1200
```

Continued

```
root id 00:00:00:00:00:00 priority 32768 ifcost 0 port 0
groups: bridge vm-switch viid-10cd3@
nd6 options=1<PERFORMNUD>
```

 and connect a new *iocage(8)* jail to it:

```
root@jailhost:~ # iocage create -r 12.1-RELEASE -n vnetcage \
  interfaces="vnet0:vm-services" ip4_addr="vnet0|10.1.1.2/24" \
  defaultrouter="10.1.1.1" vnet_default_interface="vm-services" \
  vnet=on
vnetcage successfully created!
root@jailhost:~ # iocage start vnetcage
No default gateway found for ipv6.
* Starting vnetcage
  + Started OK
  + Using devfs_ruleset: 5
  + Configuring VNET OK
  + Using IP options: vnet
  + Starting services OK
  + Executing poststart OK
root@jailhost:~ # ifconfig -g epair
vnet0.29
root@jailhost:~ # ifconfig vnet0.29
vnet0.29: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST>\
metric 0 mtu 1500
    description: associated with jail: vnetcage as nic: epair0b
    options=8<VLAN_MTU>
    ether f4:4d:30:13:1e:5c
    hwaddr 02:82:25:a0:cf:0a
    inet6 fe80::f64d:30ff:fe13:1e5c%vnet0.29 prefixlen 64 scopeid 0x4
    groups: epair
    media: Ethernet 10Gbase-T (10Gbase-T <full-duplex>)
    status: active
    nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
root@jailhost:~ # ifconfig vm-services
vm-services: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0\
mtu 1500
    ether 0a:3b:ea:3e:5c:eb
    inet 10.1.1.1 netmask 0xfffff00 broadcast 10.1.1.255
    id 00:00:00:00:00:00 priority 32768 hellotime 2 fwddelay 15
    maxage 20 holdcnt 6 proto rstp maxaddr 2000 timeout 1200
    root id 00:00:00:00:00:00 priority 32768 ifcost 0 port 0
    member: vnet0.29 flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
        ifmaxaddr 0 port 4 priority 128 path cost 2000
    groups: bridge vm-switch viid-10cd3@
    nd6 options=1<PERFORMNUD>
```

Continued

```
root@jailhost:~ # iocage console vnetcage
root@vnetcage:~ # fetch -q -o - http://canhazip.com
fetch: http://canhazip.com: No address record
root@vnetcage:~ # host canhazip.com
;; connection timed out; no servers could be reached
root@vnetcage:~ # logout
root@jailhost:~ #
```

To allow that jail to talk to the world, we'll add minimal configuration to add outbound NAT for it (this assumes *pf(4)* isn't running already):

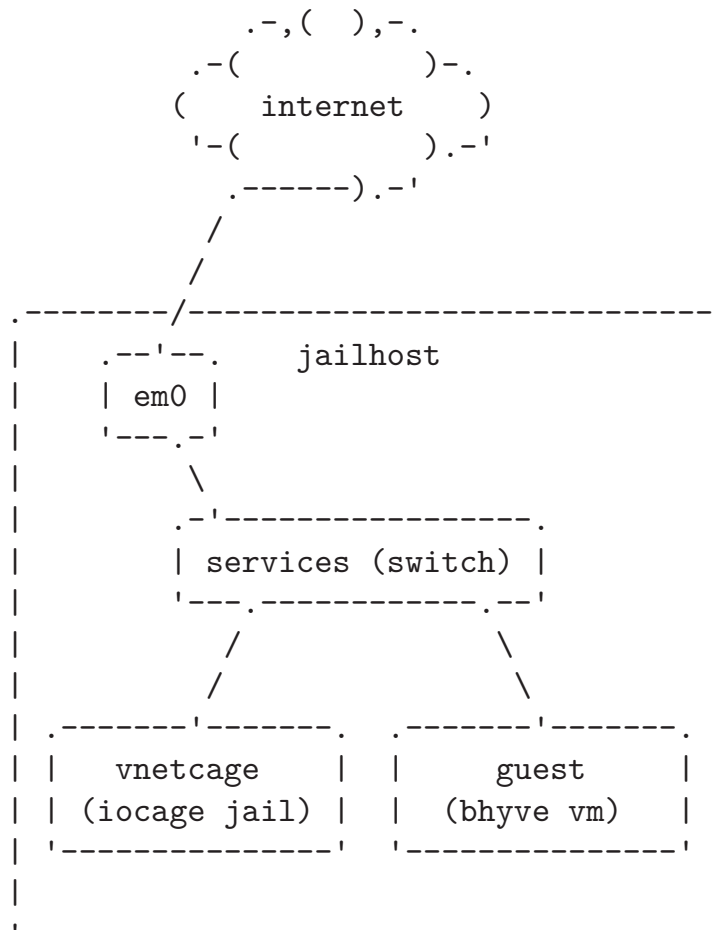
```
root@jailhost:~ # echo "set skip on lo0" >/etc/pf.conf
root@jailhost:~ # echo "nat on em0 from 10.1.1/24 -> em0:0" >>/etc/pf.conf
root@jailhost:~ # service pf enable
pf enabled in /etc/rc.conf
root@jailhost:~ # service pf start
Enabling pf.
root@jailhost:~ # iocage console vnetcage
root@vnetcage:~ # fetch -q -o - http://canhazip.com
<your public facing IP shown here>
root@vnetcage:~ # logout
root@jailhost:~ #
```


Adding bhyve VMs and DHCP to the Mix

In the example above, a virtual switch called "services" was created. It's backed by the bridge interface "vm-services" and has one VNET jail named "vnetcage" connected to it, which uses NAT to talk to the world.

In the next step, a bhyve VM running FreeBSD is added to the same virtual switch, using a different IP address on network 10.1.1.0/24.


Network diagram:





To ease the provisioning process, we make the bhyve VM get its IP address over DHCP. `dns/dnsmasq` is used for this purpose:

```
root@jailhost:~ # pkg install dnsmasq
...
cat >/usr/local/etc/dnsmasq.conf <<EOF
domain-needed
listen-address=10.1.1.1
interface=vm-services
bind-interfaces
local-service
dhcp-authoritative
dhcp-range=10.1.1.10,10.1.1.20
EOF
root@jailhost:~ # service dnsmasq enable
dnsmasq enabled in /etc/rc.conf
root@jailhost:~ # service dnsmasq start
```




We already installed `sysutils/vm-bhyve` earlier; all that's left to do is download the install ISO, create the VM, connect it to the switch, and run the installer:

```
root@jailhost:~ # vm iso https://download.freebsd.org/ftp/releases/\
ISO-IMAGES/12.1/FreeBSD-12.1-RELEASE-amd64-bootonly.iso
...
root@jailhost:~ # vm create guest
root@jailhost:~ # vm add -d network -s services guest
root@jailhost:~ # vm install guest \
  FreeBSD-12.1-RELEASE-amd64-bootonly.iso
root@jailhost:~ # vm console guest
...
root@jailhost:~ # vm stop guest
root@jailhost:~ # vm start guest
```

This uses the bootonly ISO (network based installation), which should work ok, as the IP address is assigned to the guest by `dnsmasq(8)` (which also provides DNS service) and NAT is already configured on the host firewall.

Note: This shows two network interfaces. `vtnet1` is the one to use in our setup. `vtnet0` can be removed manually by modifying the VM's configuration file `/zroot/vms/guest/guest.conf`.



As expected, `dnsmasq(8)` assigned an IP address from the DHCP pool to the new VM (10.1.1.11/24 in this example). In an environment where having a stable IP is important, it's advantageous to assign fixed IP addresses that aren't part of the dynamic pool based on the virtual network interface's MAC address in `/usr/local/etc/dnsmasq.conf`, e.g.,

```
dhcp-host=11:22:33:44:55:66,192.168.0.60
```

Once we added a non-privileged user and enabled `sshd(8)` in the new VM, we are able to `ssh(1)` into it:

```
root@jailhost:~ # ssh user@10.1.1.11
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.1.1.11' (ECDSA) to the list of known hosts.
Password for user@guest:
$ fetch -q -o - http://canhazip.com
<your public facing IP shown here>
$ ^DConnection to 10.1.1.11 closed.
root@jailhost:~ #
```

Note: By exposing `ldev/bpf` to a VNET jail, it's possible to configure the jail's IP address using DHCP just like it's done for VMs here. Using `iocage(8)` this is easily accomplished by enabling the `dhcp` property.

Preventing Traffic Between VNET Jails/VMs

A simple way to prevent traffic between jails and VMs is to set the private flag on bridge members (`ifconfig <bridgename> private <interfacename>`). Any interface marked as private won't be able to talk to any other interface that is also marked as such.

`sysutils/vm-bhyve` sets the private flag automatically on `vm start` if the switch was configured to be private beforehand by running `vm switch private <switchname> on`.

Unfortunately, this is only true for VMs, so if you connect `iocage` jails to the switch, you'll have to set the private flag for them manually on every jail start.

In the current setup, "vnetcage" can ssh into "guest":


```
root@jailhost:~ # iocage console -f vnetcage
root@vnetcage:~ # nc 10.1.1.11 22
SSH-2.0-OpenSSH_7.8 FreeBSD-20180909
^C
root@vnetcage:~ # logout
root@jailhost:~ #
```

After changing the switch to private mode, the VM's tap interface connected to the bridge is marked as PRIVATE. As the jail's epair interface is not tagged as PRIVATE, traffic can still flow and ssh still works:

```
root@jailhost:~ # vm stop guest
Sending ACPI shutdown to guest
root@jailhost:~ # vm switch private services on
root@jailhost:~ # vm switch list
NAME      TYPE      IFACE      ADDRESS      PRIVATE  MTU  VLAN  PORTS
services standard vm-services 10.1.1.1/24  yes      -    -    -
root@jailhost:~ # vm start guest
Starting guest
* found guest in /zroot/vms/guest
```



Continued

```
* booting...
root@jailhost:~ # ifconfig vm-services
vm-services: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0\
mtu 1500
    ether 0a:3b:ea:3e:5c:eb
    inet 10.1.1.1 netmask 0xffffffff broadcast 10.1.1.255
    id 00:00:00:00:00:00 priority 32768 hellotime 2 fwdelay 15
    maxage 20 holdcnt 6 proto rstp maxaddr 2000 timeout 1200
    root id 00:00:00:00:00:00 priority 32768 ifcost 0 port 0
    member: tap1 flags=943<LEARNING,DISCOVER,PRIVATE,AUTOEDGE,AUTOPTP>
        ifmaxaddr 0 port 6 priority 128 path cost 2000000
    member: vnet0.31 flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
        ifmaxaddr 0 port 7 priority 128 path cost 2000
    groups: bridge vm-switch viid-10cd3@
    nd6 options=1<PERFORMNUD>
root@jailhost:~ # iocage console -f vnetcage
root@vnetcage:~ # nc 10.1.1.11 22
SSH-2.0-OpenSSH_7.8 FreeBSD-20180909
^C
root@vnetcage:~ # logout
root@jailhost:~ #
```



Let's change the jail's connection manually to "private" and perceive that connection attempts to the jail time out, while NAT to the outside world is still intact and the jailhost can still connect to the VM via *ssh(1)*:

```
root@jailhost:~ # ifconfig vm-services private vnet0.31
root@jailhost:~ # iocage console -f vnetcage
root@vnetcage:~ # nc -vw 10 10.1.1.11 22
nc: connect to 10.1.1.11 port 22 (tcp) failed: Operation timed out
root@vnetcage:~ # fetch -q -o - http://canhazip.com
<your public facing IP shown here>
root@vnetcage:~ # logout
root@jailhost:~ # nc 10.1.1.11 22
SSH-2.0-OpenSSH_7.8 FreeBSD-20180909
^C
root@jailhost:~ #
```



Ideally, *iocage(8)* would support a configuration option to allow setting the private flag on the bridge member automatically. Until such a feature becomes available, the script below can be configured to be executed by the jail's poststart hook to accomplish (almost) the same.

/usr/local/sbin/set_ioc_vnet_private.sh:

```
#!/bin/sh

set -e

if [ "$#" -ne 1 -a "$#" -ne 2 ]; then
    echo "Usage: $0 jailname [vnetprefix]" >&2
    exit 1
fi

JAILNAME=$1
VNETPREFIX=${2:-vnet0}

JAILINFO=$(jls -j ioc-$JAILNAME)
JID=$(echo "$JAILINFO" | grep $JAILNAME | awk '{ print $1 }')

for BRIDGE in $(ifconfig -g bridge); do
    CONFIG=$(ifconfig $BRIDGE)
    set +e
    echo "$CONFIG" | grep "member: ${VNETPREFIX}\.${JID} >/dev/null
    NOTFOUND=$?
    set -e
    if [ $NOTFOUND -eq 0 ]; then
        ifconfig $BRIDGE private $VNETPREFIX.$JID
        exit 0
    fi
done

echo "Couldn't find interface $VNETPREFIX.$JID on any bridges" >&2
exit 1
```

The script takes the jail's name as a mandatory parameter and optionally the vnet interface (in iocage's internal enumeration). The latter defaults to "vnet0".

Now, let's restart the "vnetcage" jail, check the bridge member configuration, then alter its configuration to make use of the new script, restart again, and compare the resulting bridge configuration:

```
root@jailhost:~ # iocage restart vnetcage
...
root@jailhost:~ # ifconfig vm-services | grep vnet
    member: vnet0.15 flags=143<LEARNING,DISCOVER,AUTOEDGE,AUTOPTP>
root@jailhost:~ # ifconfig vm-services | grep vnet
root@jailhost:~ # iocage set \
    exec_poststart="/usr/local/sbin/set_ioc_vnet_private.sh vnetcage" \
    vnetcage
exec_poststart: /usr/bin/true -> /usr/local/sbin/set_ioc_vnet_private.sh \
    vnetcage
root@jailhost:~ # iocage restart vnetcage
...
```

Continued

```
root@jailhost:~ # ifconfig vm-services | grep vnet
    member: vnet0.16 flags=943<LEARNING,DISCOVER,PRIVATE,AUTOEDGE,\
AUTOPTP>
root@jailhost:~ #
```

As you can see, the bridge member is now correctly configured to be private on jail start.

Note: This solution isn't perfect, as traffic is possible in the short period of time between starting the jail and running the `exec_poststart` command.

Firewalling Inside VNET Jails/VMs

Firewalling inside bhyve VMs is straight-forward—simply run the host firewall of the OS running inside the VM.

Running a firewall inside a jail is a bit more complicated. `ipfw(8)` is the recommended option, but `pf(4)` should work at this point as well. For the sake of not introducing even more syntax in this article, we'll describe the latter here, even though `iocage`'s documentation recommends otherwise.

To run `pf(4)` inside a VNET jail, the `pf(4)` kernel module has to be loaded and various devices need to be exposed. This is done by adding a new ruleset set to `/etc/devfs.rules`:

```
[vnet_jail_pf=501]
add include $devfsrules_hide_all
add include $devfsrules_unhide_basic
add include $devfsrules_unhide_login
add include $devfsrules_jail
add path pf unhide
add path pflog unhide
```

and applying it to the jail:

```
root@jailhost:~ # service devfs restart
root@jailhost:~ # iocage stop vnetcage
...
root@jailhost:~ # iocage set devfs_ruleset=501 vnetcage
devfs_ruleset: 4 -> 501
root@jailhost:~ # iocage console -f vnetcage
...
root@vnetcage:~ # ls /dev/pf
/dev/pf
root@vnetcage:~ # logout
root@jailhost:~ #
```

Note: Due to a bug in `iocage(8)`, the configured `devfs(8)` ruleset is removed from `devfs(8)` every time the jail is stopped. This will probably get fixed in a future release; in the meantime there's a patch available (<https://github.com/iocage/iocage/pull/1106>) that addresses the issue.



Test using a minimal firewall configuration—check if blocking a specific IP address works and if state is created like expected:

```
root@jailhost:~ # iocage console -f vnetcage
root@vnetcage:~ # echo "set skip on lo0" >/etc/pf.conf
root@vnetcage:~ # echo "block quick to 1.1.1.1" >>/etc/pf.conf
root@vnetcage:~ # echo "pass" >>/etc/pf.conf
root@vnetcage:~ # service pf enable
pf enabled in /etc/rc.conf
root@vnetcage:~ # service pf start
Enabling pf.
root@vnetcage:~ # service pf start
root@vnetcage:~ # ping -c1 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=56 time=16.475 ms

--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 16.475/16.475/16.475/0.000 ms
root@vnetcage:~ # pfctl -s state
all icmp 10.1.1.2:39231 -> 8.8.8.8:39231      0:0
root@vnetcage:~ # ping -c1 1.1.1.1
PING 1.1.1.1 (1.1.1.1): 56 data bytes
ping: sendto: Permission denied

--- 1.1.1.1 ping statistics ---
1 packets transmitted, 0 packets received, 100.0% packet loss
root@vnetcage:~ # fetch -q -o - http://canhazip.com
93.104.68.83
root@vnetcage:~ # pfctl -s state
all icmp 10.1.1.2:39231 -> 8.8.8.8:39231      0:0
all udp 10.1.1.2:32682 -> 8.8.8.8:53          MULTIPLE:SINGLE
all tcp 10.1.1.2:58192 -> 104.16.223.38:80          FIN_WAIT_2:FIN_WAIT_2
root@vnetcage:~ # logout
root@jailhost:~ #
```

Note: Jails/VMs on the same bridge can set/steal IPs and change MAC addresses, so this level of protection is only sufficient to prevent unwanted/accidental traffic from happening. For tighter security, it's possible to enable Layer 2 filtering using ipfw(8) on the jailhost and insert an additional bridge between the host bridge and the jail's epair interface. The details of such a setup are beyond the scope of this article.

VXLAN

Jails and VMs are virtualization technologies that add a lot of flexibility and agility to the provisioning of resources.

VXLAN (Virtual eXtensible LAN interface) is a tunneling protocol that aims to decouple the virtual network from the underlying physical network and thereby ease automation and orchestration. It does so by encapsulating Layer 2 Ethernet frames into Layer 3 IP/UDP packets. One can think of VXLAN as VLAN for multi-



tenant data centers. And just like VLAN uses a unique VLAN tag, VXLAN uses a VXLAN Network Identifier (VNI), a 24-bit value in the VXLAN header, to distinguish network segments.

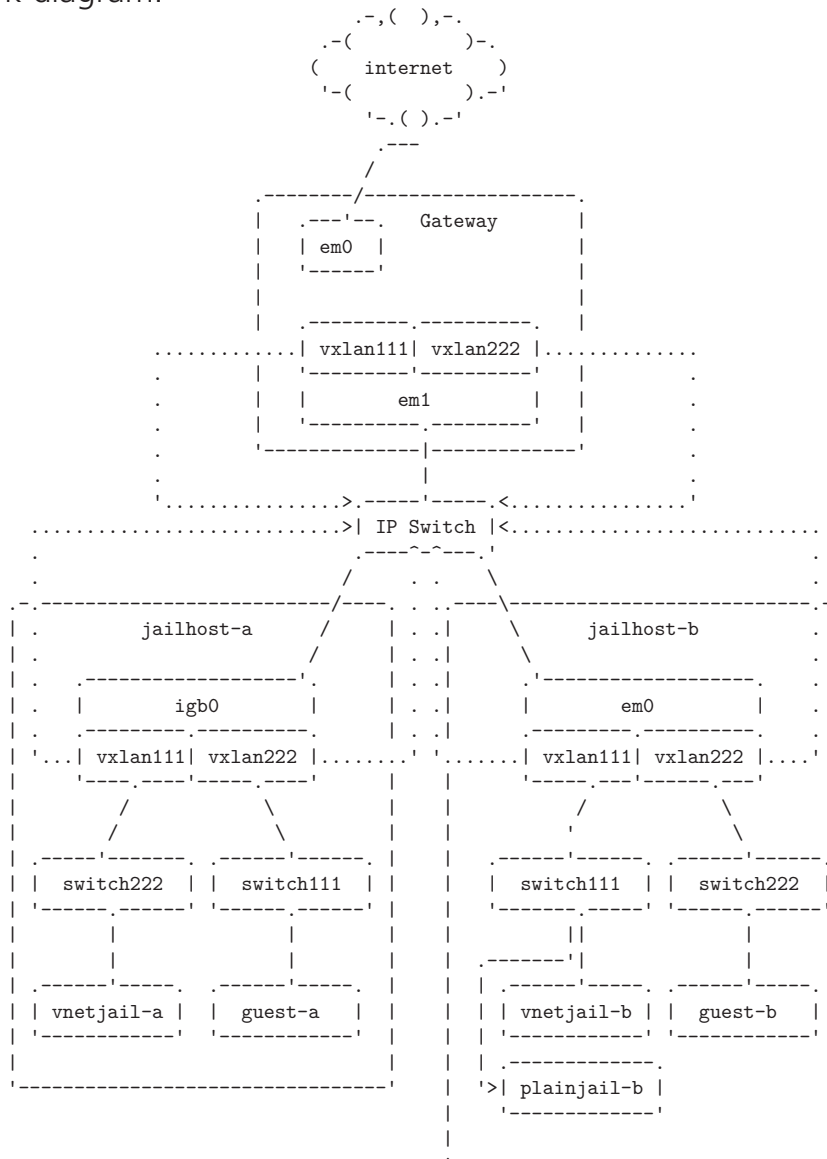
VXLAN Example Overview

The example setup consists of three hosts on the same LAN, one gateway host (LAN IP 192.168.0.1) and two jailhosts (“jailhost-a” and “jailhost-b”) with LAN IP addresses 192.168.0.10 and 192.168.0.20. The jailhosts are hosting VNET jails and VMs; host “jailhost-b” also hosts a plain jail.

The three hosts are connected over two VXLANs (VXLAN ids 111 and 222); the gateway host provides access to the internet over a dedicated uplink using NAT. Jails and VMs get IP addresses on both VXLANs (networks 10.0.111.0/24 and 10.0.222.0/24).

In this example, many configurations will be applied by rebooting the machines involved. Even though this is mostly done for the sake of brevity, it also reboots the setup as a side-effect—something that needs to be done anyway. Everything described can also be accomplished without rebooting though.

Network diagram:



The example uses multicast mode. VXLAN can also be configured in unicast mode (see *vxlan(4)* for details).

Gateway Configuration

The gateway host uses a simple *ipfw(8)/natd(8)* configuration and two *vxlan(4)* interfaces. It is assumed that the uplink is already configured:

Firewall, NAT and IP forwarding:

```
root@gateway:~ # service ipfw enable
ipfw enabled in /etc/rc.conf
root@gateway:~ # sysrc firewall_type=open
firewall_type: UNKNOWN -> open
root@gateway:~ # service natd enable
natd enabled in /etc/rc.conf
root@gateway:~ # sysrc natd_interface=em0
natd_interface: -> em0
root@gateway:~ # sysrc gateway_enable=YES
gateway_enable: NO -> YES
```

VXLAN interfaces:

```
root@gateway:~ # sysrc cloned_interfaces+="vxlan111 vxlan222"
cloned_interfaces: -> vxlan111 vxlan222
root@gateway:~ # sysrc ifconfig_vxlan111="inet 10.0.111.1/24 mtu 1450"
ifconfig_vxlan111: -> inet 10.0.111.1/24 mtu 1450
root@gateway:~ # sysrc create_args_vxlan111="vxlanid 111 vxlanlocal\
192.168.0.1 vxlandev em1 vxlangroup 239.0.0.111"
create_args_vxlan111: -> vxlanid 111 vxlanlocal 192.168.0.1
vxlandev em1 vxlangroup 239.0.0.111
root@gateway:~ # sysrc ifconfig_vxlan222="inet 10.0.222.1/24 mtu 1450"
ifconfig_vxlan222: -> inet 10.0.222.1/24 mtu 1450
root@gateway:~ # sysrc create_args_vxlan222="vxlanid 222 vxlanlocal\
192.168.0.1 vxlandev em1 vxlangroup 239.0.0.222"
create_args_vxlan222: -> vxlanid 222 vxlanlocal 192.168.0.1
vxlandev em1 vxlangroup 239.0.0.222
```

Static routes to VXLAN multicast addresses:

```
root@gateway:~ # sysrc static_routes+="vxlan111 vxlan222"
static_routes: -> vxlan111 vxlan222
root@gateway:~ # sysrc route_vxlan111="239.0.0.111/32 -interface em1"
route_vxlan111: -> 239.0.0.111/32 -interface em1
root@gateway:~ # sysrc route_vxlan222="239.0.0.222/32 -interface em1"
route_vxlan222: -> 239.0.0.222/32 -interface em1
root@gateway:~ # reboot
```

Jailhost-a

“jailhost-a” hosts one VM and one VNET jail, connected over switches (bridge interfaces) to the VXLAN interfaces, which in turn use the physical interface *igb0* to transmit the encapsulated traffic over the LAN.



Network Configuration (jailhost-a)

Create VXLAN interfaces:

```
root@jailhost-a:~ # sysrc cloned_interfaces+="vxlan111 vxlan222"
cloned_interfaces:  -> vxlan111 vxlan222
root@jailhost-a:~ # sysrc ifconfig_vxlan111="inet 10.0.111.10/24 mtu 1450"
ifconfig_vxlan111:  -> inet 10.0.111.10/24 mtu 1450
root@jailhost-a:~ # sysrc create_args_vxlan111="vxlanid 111 vxlanlocal\
 192.168.0.10 vxlandev igb0 vxlangroup 239.0.0.111"
create_args_vxlan111:  -> vxlanid 111 vxlanlocal 192.168.0.10
  vxlandev igb0 vxlangroup 239.0.0.111
root@jailhost-a:~ # sysrc ifconfig_vxlan222="inet 10.0.222.10/24 mtu 1450"
ifconfig_vxlan222:  -> inet 10.0.222.10/24 mtu 1450
root@jailhost-a:~ # sysrc create_args_vxlan222="vxlanid 222 vxlanlocal\
 192.168.0.10 vxlandev igb0 vxlangroup 239.0.0.222"
create_args_vxlan222:  -> vxlanid 222 vxlanlocal 192.168.0.10
  vxlandev igb0 vxlangroup 239.0.0.222
```



Set static routes for multicast traffic (technically not needed if the default route goes through the same interface):

```
root@jailhost-a:~ # sysrc static_routes+="vxlan111 vxlan222"
static_routes:  -> vxlan111 vxlan222
root@jailhost-a:~ # sysrc route_vxlan111="239.0.0.111/32 -interface igb0"
route_vxlan111:  -> 239.0.0.111/32 -interface igb0
root@jailhost-a:~ # sysrc route_vxlan222="239.0.0.222/32 -interface igb0"
route_vxlan222:  -> 239.0.0.222/32 -interface igb0
root@jailhost-a:~ # reboot
```



Create switches (bridge interfaces) to connect jails/VMs to and add the respective VXLAN interfaces to them:

```
root@jailhost-a:~ # sysrc cloned_interfaces+="bridge0 bridge1"
cloned_interfaces: vxlan111 vxlan222 -> vxlan111 vxlan222 bridge0 bridge1
root@jailhost-a:~ # sysrc ifconfig_bridge0_name="switch111"
ifconfig_bridge0_name:  -> switch111
root@jailhost-a:~ # sysrc \
  ifconfig_switch111="inet 10.0.111.12/32 addm vxlan111"
ifconfig_switch111:  -> inet 10.0.111.12/32 addm vxlan111
root@jailhost-a:~ # sysrc ifconfig_bridge1_name="switch222"
ifconfig_bridge1_name:  -> switch222
root@jailhost-a:~ # sysrc \
  ifconfig_switch222="inet 10.0.222.12/32 addm vxlan222"
ifconfig_switch222:  -> inet 10.0.222.12/32 addm vxlan222
root@jailhost-a:~ # reboot
```



VM Configuration (jailhost-a)

Install `sysutils/vm-bhyve` and create a switch of type "manual" (not managed by

vm-bhyve) that refers to the bridge interface (switch111) created above:

```
root@jailhost-a:~ # pkg install vm-bhyve
...
root@jailhost-a:~ # service vm enable
vm enabled in /etc/rc.conf
root@jailhost-a:~ # sysrc vm_dir=zfs:zroot/vms
vm_dir: -> zfs:zroot/vms
root@jailhost-a:~ # zfs create zroot/vms
root@jailhost-a:~ # vm init
root@jailhost-a:~ # vm switch create -t manual -b switch111 switch111
root@jailhost-a:~ # vm switch list
```

NAME	TYPE	IFACE	ADDRESS	PRIVATE	MTU	VLAN	PORTS
switch111	manual	switch111	n/a	no	n/a	n/a	n/a

Download FreeBSD ISO and install a VM called "guest-a"; start new VM on boot:

```
root@jailhost-a:~ # vm iso https://download.freebsd.org/ftp/releases/\
ISO-IMAGES/12.1/FreeBSD-12.1-RELEASE-amd64-bootonly.iso
root@jailhost-a:~ # vm create guest-a
root@jailhost-a:~ # vm add -d network -s switch111 guest-a
root@jailhost-a:~ # vm install \
  guest-a FreeBSD-12.1-RELEASE-amd64-bootonly.iso
root@jailhost-a:~ # vm console guest-a
... (set IP of vtnet1 to 10.0.111.13/24, set default gw to 10.0.111.1)
root@jailhost-a:~ # sysrc vm_list+="guest-a"
root@jailhost-a:~ # vm stop guest-a
```

Start the VM and test connectivity (note how public traffic is sent over VXLAN 111 and NATted at the gateway host, as the gateway host is configured to be the default gateway):

```
root@jailhost-a:~ # vm start guest-a
root@jailhost-a:~ # vm console guest-a
root@guest-a:~ # ping -c 3 10.0.111.1
PING 10.0.111.1 (10.0.111.1): 56 data bytes
64 bytes from 10.0.111.1: icmp_seq=0 ttl=64 time=1.545 ms
64 bytes from 10.0.111.1: icmp_seq=1 ttl=64 time=0.793 ms
64 bytes from 10.0.111.1: icmp_seq=2 ttl=64 time=0.790 ms

--- 10.0.111.1 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.790/1.043/1.545/0.355 ms
root@guest-a:~ # traceroute www.freebsd.org
 1 10.0.111.1 (10.0.111.1) 1.251 ms 1.421 ms 1.070 ms
...
root@guest-a:~ # logout
$ ^D
```




Jail Configuration (jailhost-a)

Install `sysutils/iocage`:


```
root@jailhost-a:~ # pkg install py36-iocage
root@jailhost-a:~ # iocage activate zroot
ZFS pool 'zroot' successfully activated.
root@jailhost-a:~ # mount -t fdescfs null /dev/fd
root@jailhost-a:~ # service iocage enable
iocage enabled in /etc/rc.conf
```

Note: You might want to mount `fdescfs(5)` permanently by altering `/etc/fstab`.



Create jail named “vnetjail-a” on VXLAN 222 (configured to be started at boot time, so it will start immediately after creation):

```
root@jailhost-a:~ # iocage create -n vnetjail-a -r 12.1-RELEASE \
  interfaces="vnet0:switch222" ip4_addr="vnet0|10.0.222.13/24" \
  boot=1 vnet_default_interface="switch222" defaultrouter="10.0.222.1" \
  vnet=on
...
vnetjail-a successfully created!
```



Enter the jail and test connectivity:

```
root@jailhost-a:~ # iocage console vnetjail-a
root@vnetjail-a:~ # traceroute -n 141.1.1.1
traceroute to 141.1.1.1 (141.1.1.1), 64 hops max, 40 byte packets
 1  10.0.111.1 (10.0.111.1)  1.353 ms  0.687 ms  1.045 ms
 2  many more interesting hosts...
...
root@vnetjail-a:~ # fetch -q -o - http://canhazip.com
(your public IP here)
root@vnetjail-a:~ # logout
```



Finally, test if everything comes up correctly on reboot:

```
root@jailhost-a:~ # reboot
```



Jailhost-b

The setup of “jailhost-b” is similar to that of “jailhost-a”, but with networks reversed (VM on VXLAN 222, jails on VXLAN 111). In addition, it hosts a plain jail that uses an alias address directly on the VXLAN interface (`vlan111`), which doesn’t use that network as its default gateway. In a production setup, this jail could be used to contain a supporting service provided by the underlying jailhost.

Network Configuration (jailhost-b)

Create VXLAN interfaces:

```
root@jailhost-b:~ # sysrc cloned_interfaces+="vxlan111 vxlan222"
cloned_interfaces: -> vxlan111 vxlan222
root@jailhost-b:~ # sysrc ifconfig_vxlan111="inet 10.0.111.20/24 mtu 1450"
ifconfig_vxlan111: -> inet 10.0.111.20/24 mtu 1450
root@jailhost-b:~ # sysrc create_args_vxlan111="vxlanid 111 vxlanlocal\
192.168.0.20 vxlandev em0 vxlangroup 239.0.0.111"
create_args_vxlan111: -> vxlanid 111 vxlanlocal 192.168.0.20
vxlandev em0 vxlangroup 239.0.0.111
root@jailhost-b:~ # sysrc ifconfig_vxlan222="inet 10.0.222.20/24 mtu 1450"
ifconfig_vxlan222: -> inet 10.0.222.20/24 mtu 1450
root@jailhost-b:~ # sysrc create_args_vxlan222="vxlanid 222 vxlanlocal\
192.168.0.20 vxlandev em0 vxlangroup 239.0.0.222"
create_args_vxlan222: -> vxlanid 222 vxlanlocal 192.168.0.20
vxlandev em0 vxlangroup 239.0.0.222
```

Set static routes for multicast traffic (technically not needed if the default route goes through the same interface):

```
root@jailhost-b:~ # sysrc static_routes+="vxlan111 vxlan222"
static_routes: -> vxlan111 vxlan222
root@jailhost-b:~ # sysrc route_vxlan111="239.0.0.111/32 -interface em0"
route_vxlan111: -> 239.0.0.111/32 -interface em0
root@jailhost-b:~ # sysrc route_vxlan222="239.0.0.222/32 -interface em0"
route_vxlan222: -> 239.0.0.222/32 -interface em0
root@jailhost-b:~ # reboot
```

Plain Jail Configuration (jailhost-b)

Install *sysutils/iocage*:

```
root@jailhost-b:~ # pkg install py36-iocage
...
root@jailhost-b:~ # iocage activate zroot
ZFS pool 'zroot' successfully activated.
root@jailhost-b:~ # mount -t fdescfs null /dev/fd
root@jailhost-b:~ # service iocage enable
iocage enabled in /etc/rc.conf
```

*Note: You might want to mount *fdescfs(5)* permanently by altering */etc/fstab*.*

Create a plain (*non-VNET*) jail on the alias address 10.0.111.21. It's configured to run at boot time, so it starts immediately after creation:

```
root@jailhost-b:~ # iocage create -n plainjail-b \
-r 12.1-RELEASE ip4_addr="vxlan111|10.0.111.21/32" \
allow_raw_sockets=1 boot=1
plainjail-b successfully created!
```

Run jail and test connectivity:

```
root@jailhost-b:~ # iocage console -f plainjail-b
...
root@plainjail-b:~ # ping -c 3 10.0.111.1
PING 10.0.111.1 (10.0.111.1): 56 data bytes
64 bytes from 10.0.111.1: icmp_seq=0 ttl=64 time=0.811 ms
64 bytes from 10.0.111.1: icmp_seq=1 ttl=64 time=0.816 ms
64 bytes from 10.0.111.1: icmp_seq=2 ttl=64 time=0.810 ms

--- 10.0.111.1 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.810/0.812/0.816/0.003 ms
root@plainjail-b:~ # ping -c 3 10.0.111.10
PING 10.0.111.10 (10.0.111.10): 56 data bytes
64 bytes from 10.0.111.10: icmp_seq=0 ttl=64 time=0.716 ms
64 bytes from 10.0.111.10: icmp_seq=1 ttl=64 time=0.297 ms
64 bytes from 10.0.111.10: icmp_seq=2 ttl=64 time=0.319 ms

--- 10.0.111.10 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.297/0.444/0.716/0.193 ms
root@plainjail-b:~ # logout
root@jailhost-b:~ #
```

Note: The way "plainjail-b" is configured, its public traffic won't get sent over VXLAN and NATted by the gateway host. That's intentional in this setup; otherwise a VNET jail would've been used.

Network Switch Setup (jailhost-b)

Create switched (bridge interfaces) to connect jails/VMs to and add the respective VXLAN interfaces to them:

```
root@jailhost-b:~ # sysrc cloned_interfaces+="bridge0 bridge1"
cloned_interfaces: vxlan111 vxlan222 -> vxlan111 vxlan222 bridge0 bridge1
root@jailhost-b:~ # sysrc ifconfig_bridge0_name="switch111"
ifconfig_bridge0_name: -> switch111
root@jailhost-b:~ # sysrc \
  ifconfig_switch111="inet 10.0.111.22/32 addm vxlan111"
ifconfig_switch111: -> inet 10.0.111.22/32 addm vxlan111
root@jailhost-b:~ # sysrc ifconfig_bridge1_name="switch222"
ifconfig_bridge1_name: -> switch222
root@jailhost-b:~ # sysrc \
  ifconfig_switch222="inet 10.0.222.22/32 addm vxlan222"
ifconfig_switch222: -> inet 10.0.222.22/32 addm vxlan222
root@jailhost-b:~ # reboot
```

VNET Jail Configuration (jailhost-b)

Create jail named "vnetjail-b" on VXLAN 111 (configured to be started at boot time,



so it'll start immediately after creation):

```
root@jailhost-b:~ # iocage create -n vnetjail-b -r 12.1-RELEASE \  
  interfaces="vnet0:switch111" ip4_addr="vnet0|10.0.111.23/24" \  
  boot=1 vnet_default_interface="switch111" defaultrouter="10.0.111.1" \  
  vnet=on  
vnetjail-b successfully created!
```

Enter the jail and test connectivity:

```
root@jailhost-b:~ # iocage console vnetjail-b  
root@vnetjail-b:~ # traceroute -n 141.1.1.1  
traceroute to 141.1.1.1 (141.1.1.1), 64 hops max, 40 byte packets  
 1  10.0.111.1 (10.0.111.1)  1.353 ms  0.687 ms  1.045 ms  
 2  many more interesting hosts...  
...  
root@vnetjail-b:~ # logout
```

VM Configuration (jailhost-b)

Install *sysutils/vm-bhyve* and create a switch of type "manual" (not managed by *vm-bhyve*) that refers to the bridge interface (switch111) created above:

```
root@jailhost-b:~ # pkg install vm-bhyve  
...  
root@jailhost-b:~ # service vm enable  
vm enabled in /etc/rc.conf  
root@jailhost-b:~ # sysrc vm_dir=zfs:zroot/vms  
vm_dir:  -> zfs:zroot/vms  
root@jailhost-b:~ # zfs create zroot/vms  
root@jailhost-b:~ # vm init  
root@jailhost-b:~ # vm switch create -t manual -b switch222 switch222  
root@jailhost-b:~ # vm switch list  
NAME          TYPE      IFACE      ADDRESS  PRIVATE  MTU  VLAN  PORTS  
switch222    manual   switch222  n/a      no       n/a  n/a   n/a
```

Download FreeBSD ISO and install a VM called "guest-b"; start new VM on boot:

```
root@jailhost-b:~ # vm iso https://download.freebsd.org/ftp/releases/  
ISO-IMAGES/12.1/FreeBSD-12.1-RELEASE-amd64-bootonly.iso  
root@jailhost-b:~ # vm create guest-b  
root@jailhost-b:~ # vm add -d network -s switch222 guest-b  
root@jailhost-b:~ # vm install \  
  guest-b FreeBSD-12.1-RELEASE-amd64-bootonly.iso  
root@jailhost-b:~ # vm console guest-b  
... (set IP of vtnet1 to 10.0.222.23/24, set default gw to 10.0.222.1)  
root@jailhost-b:~ # sysrc vm_list+="guest-b"  
root@jailhost-b:~ # vm stop guest-b
```



Start the VM and test connectivity (note how public traffic is sent over VXLAN 222 and NATted at the gateway host, as the gateway host is configured to be the default gateway):

```
root@jailhost-b:~ # vm start guest-b
root@jailhost-b:~ # vm console guest-b
root@guest-b:~ # traceroute www.freebsd.org
 1  10.0.111.1 (10.0.111.1)  1.251 ms  1.421 ms  1.070 ms
...
root@guest-b:~ # logout
$ ^D
```

Finally, test if everything comes up correctly on reboot:

```
root@jailhost-b:~ # reboot
```

VXLAN Multicast Troubleshooting

There are various tools that can help to troubleshoot your VXLAN setup.

Use *ifconfig(8)* to make sure the interface is actually configured correctly:

```
root@jailhost-b:~ # ifconfig vxlan111
vxlan111: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST>
metric 0 mtu 1450
    options=80000<LINKSTATE>
    ether 58:9c:fc:10:ff:fe
    inet 10.0.111.20 netmask 0xfffff00 broadcast 10.0.111.255
    inet 10.0.111.21 netmask 0xffffffff broadcast 10.0.111.21
    groups: vxlan
    vxlan vni 111 local 192.168.0.20:4789 group 239.0.0.111:4789
    media: Ethernet autoselect (autoselect <full-duplex>)
    status: active
    nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
```

Note: The MTU is set to 1450 (as VXLAN uses 50 bytes of header information). If supported by all components, it's recommended to configure jumbo frames on your network.

Use *sockstat(1)* to make sure that the host is actually listening on the standard VXLAN port 4789:

```
root@jailhost-b:~ # sockstat -l4 | grep 4789
?      ?      ?      ?      udp4      *:4789      **
```

Use *ifmcstat(8)* to verify that the multicast configuration looks reasonable:

```
root@jailhost-b:~ # ifmcstat -i em0
em0:
    inet 192.168.0.20
```

Continued

```
igmpv3 rv 2 qi 125 qri 100 uri 3
  group 239.0.0.222 mode exclude
    mcast-macaddr 01:00:5e:00:00:de
  group 239.0.0.111 mode exclude
    mcast-macaddr 01:00:5e:00:00:6f
  group 224.0.0.1 mode exclude
    mcast-macaddr 01:00:5e:00:00:01
```

Use *tcpdump(1)* to inspect VXLAN traffic:

```
root@jailhost-b:~ # tcpdump -vni em0 -f "udp && port 4789"
tcpdump: listening on em0, link-type EN10MB (Ethernet), capture size\
262144 bytes
14:24:13.288186 IP (tos 0x0, ttl 64, id 9404, offset 0, flags [none],\
proto UDP (17), length 78)
    192.168.0.20.17657 > 239.0.0.111.4789: VXLAN, flags [I] (0x08), vni 111
ARP, Ethernet (len 6), IPv4 (len 4), Request who-has 10.0.111.1 tell\
10.0.111.20, length 28
...
```

Check the ARP table using *arp(8)* (also within VMs):

```
root@jailhost-b:~ # arp -an
? (10.0.222.22) at 02:0e:35:b7:6d:01 on switch222 permanent [bridge]
? (10.0.111.22) at 02:0e:35:b7:6d:00 on switch111 permanent [bridge]
? (10.0.222.20) at 58:9c:fc:10:ff:81 on vxlan222 permanent [ethernet]
? (10.0.111.21) at 58:9c:fc:10:ff:fe on vxlan111 permanent [ethernet]
? (10.0.111.20) at 58:9c:fc:10:ff:fe on vxlan111 permanent [ethernet]
? (10.0.111.12) at 02:f7:16:28:83:00 on vxlan111 expires in 1160 seconds\
[ethernet]
...
```

Check VXLAN forwarding tables using *sysctl(8)*:

```
root@jailhost-b:~ # sysctl net.link.vxlan.111.ftable.dump
net.link.vxlan.111.ftable.dump:
D 0x01 02:F7:16:28:83:00    192.168.0.10 00002343
D 0x01 58:9C:FC:10:FF:E9    192.168.0.10 00002378
D 0x01 00:BD:0B:07:F7:01    192.168.0.10 00002299
D 0x01 58:9C:FC:03:25:35    192.168.0.10 00002347
...
root@jailhost-b:~ # sysctl net.link.vxlan.222.ftable.dump
...
```

Flush ARP and VXLAN forwarding tables:

```
root@jailhost-b:~ # arp -ad
10.0.111.12 (10.0.111.12) deleted
10.0.111.10 (10.0.111.10) deleted
192.168.0.10 (192.168.0.10) deleted
root@jailhost-b:~ # ifconfig vxlan111 vxlanflush
root@jailhost-b:~ # ifconfig vxlan222 vxlanflush
```

If the environment permits, temporarily disable host firewalls while troubleshooting to make sure they're not interfering with traffic.

Conclusion and Further Reading

The intent of this article was to show various ways to configure the network of FreeBSD's virtualization features based on examples. The choice to use third party tools from ports/packages was made consciously, as many users will try these tools in practice to get started. Even if specific tools won't make it to a production setup, using them while prototyping makes it easier to experiment and perceive what's configured before building a leaner solution.

The examples shown are by no means exhaustive, but they should help the reader to get started and figure out which kind of setup might meet their specific requirements. It's recommended to do some further reading to fully understand the available options and their implications. ●

Some good sources of topic information

Michael W Lucas — *FreeBSD Mastery: Jails* (<https://mwl.io/nonfiction/os#fmjail>)

John Nielsen — Using VXLAN to network virtual machines, jails, and other fun things on FreeBSD (<https://www.bsdcan.org/2016/schedule/events/715.en.html>)

Slides (https://www.bsdcan.org/2016/schedule/attachments/341_VXLAN_BSDCan2016.pdf)

Video (https://www.youtube.com/watch?v=_1Ne_TgF3MQ)

iocage: A FreeBSD Jail Manager (<https://iocage.readthedocs.io>)

vm-bhyve: Shell based, minimal dependency bhyve manager (<https://github.com/churchers/vm-bhyve>)

pot: another container framework for FreeBSD, based on jails, ZFS and pf (<https://github.com/pizzamig/pot>)

FreeBSD as a Host with bhyve (<https://www.freebsd.org/doc/handbook/virtualization-host-bhyve.html>), covers setting up bhyve by directly using LAN addresses.

FreeBSD *ifconfig(8)* man page (<https://www.freebsd.org/cgi/man.cgi?query=ifconfig&sektion=8>)

FreeBSD *bridge(4)* man page (<https://www.freebsd.org/cgi/man.cgi?query=bridge&sektion=4>)

FreeBSD *vxlan(4)* man page (<https://www.freebsd.org/cgi/man.cgi?query=vxlan&sektion=4>)

Growing up in the Bavarian countryside, **Michael Gmelin** started using FreeBSD at a rural userfriendly.org-style ISP in the 1990s. Spending most of the aughts developing software to automate ISP and network operator processes, he started contributing to the Project close to the end of the decade. While he moved on to build payment systems in the emerging fintech industry, he finally became a FreeBSD committer in 2014. Having written software in many different languages, he recently developed an affection for Rust. In his spare time, he likes to play classic video games and run demoscene productions. When away from the keyboard, he enjoys (making) music, cycling, woodworking, and cooking. Perl will always have a place in his heart.